A k-hyperplane-based Neural Network For Non-Linear Regression

Hongmei He *Member, IEEE* and Zengchang Qin

***Abstract*—For the time series prediction problem, the relationship between the abstracted independent variables and the response variable is usually strong non-linear. We propose a neural network fusion model based on k-hyperplanes for non-linear regression. A k-hyperplane clustering algorithm is developed to split the data to several clusters. The experiments are done on an artificial time series, and the convergence of k-hyperplane clustering algorithm and neural network gradient training algorithm is examined. The dimension of inputs affect the clustering performance very much. Neural network fusion can get some compensation in performance. It is shown that the prediction performance of the model for the time series is very good. The model can be further exploited for many real applications.**

Keyword k-hyperplane clustering algorithm, Neural network fusion model, Gradient descent learning, Non-linear regression.

I. Introduction

Most widely known prediction tools use linear regression models. The purpose of regression analysis is to model and analyze the relationship between a response variable and one or more independent variables, thus future state of the response variable can be predicted through the created model. For constructing models, a least-squares technologies is usually adopted to find the regression coefficients using the collected observations of input and output variables. The estimation process of dependent variables by statistical regression analysis is carried out based on input variables that take numeric values. However, for real-world problems, the observations are usually uncertain and imprecise, due to epistemic difference of people and/or random measurement errors. The relationship between the independent variables and the response variable is usually strongly non-linear. This requires an integrated treatment of uncertainty when modeling regression.

For regression problems, three types of approaches are generally used [1]: (1) linear programming [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], least-squares [14], [15], [16], [17], [18], [19], [20], and support vector machine [21], [22], [23], [24]. In a latest research literature, Chen and Hsueh[1] proposed a fuzzy regression approach that uses the least-squares method to minimize the total estimation error of the distance between the observed and estimated fuzzy responses. Another approach is to split a large task into several small tasks, each of which is completed by using single regression analysis. For example, cluster-wise fuzzy regression analysis was investigated in [26], [25]. In [25], minimization of particular objective functions yields simultaneous estimates for the parameters of c-regression models, together with a fuzzy c-partitioning of the data. In [26], authors apply fuzzy clustering techniques into the fuzzy regression model fitting at each step of procedure. We propose a neural network fusion model based on k-hyperplanes that represent a set of linear functions between independent variables and the response variable. A novel k-hyperplane clustering algorithm is developed to split the training data into several clusters. The estimate of a sample point can be obtained through the neural network fusion. The experiment is done on an artificial time series. We examine the convergence of the clustering algorithm and the neural network training, respectively, and compare the results of nonlinear regression with two input history variables to that with three history variables.

H. He is with the Department of Engineering Mathematics, University of Bristol, Bristol, UK, e-mail: h.he@bristol.ac.uk

Z. Qin is with Intelligent Computing and Machine Learning Lab, School of Automation and Electrical Engineering, Beihang University, Beijing, 100191 email: zcqin@buaa.edu.cn

II. The k-hyperplane clustering algorithm

There are many clustering algorithms for solving different problems, such as k-means, FCM, etc. The simplest approach is the k-means algorithm. It partitions the points in data set into k clusters. This iterative partitioning minimizes the sum of the within-cluster sums of point-to-cluster-centroid distances over all clusters. Given a data set $\mathcal{S} = \{p_1, ..., p_n\}$, $p_i = (\mathbf{x}_i, y_i)$, which is partitioned into k clusters, satisfying the conditions:

- $\mathcal{S} = \cup_{i=1}^{k} S_i$,
- $S_i \cap S_j = \phi$, and
- $S_i \neq \phi$.

For multiple regression, the purpose is to make all points in a cluster fall into a hyperplane. However, due to the uncertainty and nonlinear relationship between independent variables and the dependent variable, we need to find a hyperplane, to which, all points are as close as possible, so that the estimates are closer to the real measures. Therefore, to find such a hyperplane, the criteria is set to the distance from a point to a hyperplane. Given the j-th hyperplane $y = a_{j0} + a_{j1}x_1 + a_{j2}x_2 + ... + a_{jn}x_n$, and a point $p_i = (x_{i1}, x_{i2}, ...x_{in}, y_i)$, the distance d_{ij} from p_i to the j-th hyperplane can be calculated by:

$$d_{ij} = \frac{|a_{j1}x_{i1} + ... + a_{jn}x_{in} - y_i + a_{j0}|}{\sqrt{a_{j1}^2 + ... + a_{jn}^2 + 1}} \tag{1}$$

The k-plane clustering algorithm is subject to the minimal sum of the within-cluster sums of point-to-hyperplane distances over all clusters:

$$J = \sum_{j=1}^{k} \sum_{p_i \in D_C} d_{ij} \tag{2}$$

Initially, k clusters are produced randomly, and correspondingly k hyperplanes are produced based on the k clusters. Then all points in the data set will be relocated. The distances from each point in the data set D to all hyperplanes are calculated, and a point will belong to the hyperplane, to which, the distance from the point is minimum. Repeat the process until the change of distance sum J is less than a positive threshold α (e.g. $\alpha < 0.05$).

Algorithm 1 shows the pseudocode of the new clustering algorithm.

Algorithm 1 The k-hyperplane clustering algorithm

```
1:  Randomly produce $k$ clusters $\{C_1, ..., C_k\}$;
2:  $J = 0$;
3:  for $c = 1 : k$ do
4:      produce a hyperplane $hp_c$;
5:      $J_c = \sum_{p_i \in c} d_{ic}$;
6:      $J = J + J_c$;
7:  end for
8:  $J_{old} = J$; $J = J + \alpha + 1$;
9:  while ($|J_{old} - J| > \alpha$) do
10:     for $i = 1 : n$ do
11:         for $j = 1 : k$ do
12:             calculate $d_{ij}$;
13:         end for
            $idx_i = argmin_{j=1:k}(d_{ij})$;
14:     end for
15:     $J = 0$;
16:     for $c = 1 : k$ do
17:         produce a hyperplane $hp_c$;
18:         $J_c = \sum_{p_i \in c} d_{ic}$;
19:         $J = J + J_c$;
20:     end for
21: end while
```

III. The neural network model for information fusion

A. The neural network model

The neural network is based on the k-estimates $\tilde{y}_1$ to $\tilde{y}_k$ obtained by the k linear regression equations $f_i(\mathbf{x})$, $i = 1, ..., k$, each of which represents the mapping function between independent variable and the dependent variable for the corresponding cluster. Fig. 1 shows the structure of the neural network.

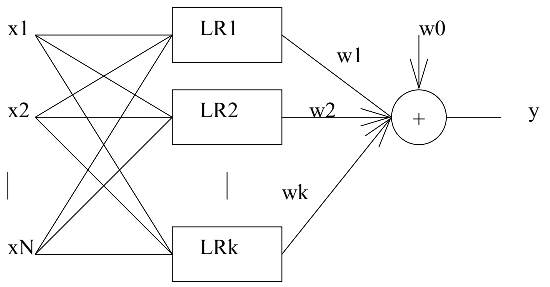


Fig. 1. The structure of the neural network model based on k-hyperplanes

Assuming we have obtained k clusters by the k-hyperplane algorithm. For each cluster C_j in the training data set, we can determine a hyperplane $\mathcal{P}_j$ in the $(n+1)$-dimension space. For each sample point p_i in the training data set, we can obtain an approximate value τ_{ij} with j-th linear function. Therefore, we have output matrix:

$$T = \begin{pmatrix} \tau_{11} & \tau_{12} & ... & \tau_{1k} \\ \tau_{21} & \tau_{22} & ... & \tau_{2k} \\ ... & ... & ... & ... \\ \tau_{N1} & \tau_{N2} & ... & \tau_{Nk} \end{pmatrix}$$

Hence, the approximation function from the neural network is of the form:

$$\tilde{y} = F(\mathbf{x_i}) = \sum_{j=1}^{k} w_j f_j(\mathbf{x_i}) + w_0. \tag{3}$$

Define $h_{0j} = 1$, $j = 1, ..., k$, and $h_{ij} = \tau_{ij}$, $i = 1, ..N$, $j = 1, ..k.$, then we have:

$$\tilde{y} = F(\mathbf{x_i}) = \sum_{j=0}^{k} w_j h_{ij}. \tag{4}$$

Hence, for N samples and k hyperplanes, we have matrix H:

$$H = \begin{pmatrix} 1, h_{11} & h_{12} & ... & h_{1k} \\ 1, h_{21} & h_{22} & ... & h_{2k} \\ ... & ... & ... & ... \\ 1, h_{N1} & h_{N2} & ... & h_{Nk} \end{pmatrix}$$

The final estimate of the dependent variable y is:

$$\tilde{y} = W \times H'. \tag{5}$$

where, $W = \{w_0, ..., w_k\}$

B. Gradient descent learning

Gradient descent learning approach calculates the gradient of the cost function only evaluated on a single training example. The parameters are then adjusted by an amount proportional to this approximate gradient. Therefore, the parameters of the model are updated after each training example. Thus the network arrives to a stable state with a local minimum of the cost function.

Assume database is partitioned into k clusters, where $1 < k < N$. Each cluster corresponds to a hyperplane. With multiple linear regression, we can obtain a linear equation for each hyperplane.

For each sample, we can have k estimates obtained by the k linear equations. According to Equation (4), for N samples, we define the cost function E as sum of squared errors of estimates and real values as below:

$$E = \frac{1}{2}\sum_{i=1}^{N}((\tilde{y_i} - y_i)^2 = \frac{1}{2}\sum_{i=1}^{N}(\sum_{j=0}^{k} w_j h_{ij} - y_i)^2 \tag{6}$$

Takefuji and Szu [27] has proved that the convergence of a neural network does not depend on the model. As long as the output v_i is the continuous, differentiable and monotonous increasing function of input u_i, namely, there exists the relationship between outputs and inputs of neurons $\frac{dv_i}{du_i} > 0$, the neural network always converges with a negative grade. Finally, the neural network arrives at a stable state with $\frac{dE}{dt} = 0$. Here we define the activation function as below:

$$v_i = \begin{cases} 1 & u_i \geq 1 \\ u_i & 0 \leq u_i < 1 \\ -1 & u_i < -1. \end{cases} \quad (7)$$

The weight in each cell indicates the state of each neuron (i.e. the output of each neuron). As defined in the activation function (7), the output v_i and input u_i of each neuron have the relationship v_i=u_i, if $u_i \in [-1, 1]$. Therefore, we have:

$$\Delta u = \Delta w = -\frac{\partial E}{\partial w_i} \times \Delta t. \quad (8)$$

Assuming updating time $\Delta t = 1$, then we have:

$$\Delta w_j = \gamma(\sum_{j=0}^{k} w_j h_{ij} - y_i)h_{ij} = \sum_{i=1}^{N}(\tilde{y}_i - y_i)h_{ij}. \quad (9)$$

where γ is the learning factor. Given a training sample $\mathbf{x}$, we can calculate the value of Equation (9), which will be as a correction of a weight. Initially, w is set to be a real value in [-1,1] randomly. The gradient descent training algorithm is different from the normal gradient algorithm that is only evaluated on a single training example in each step. Our gradient algorithm is evaluated with the whole cost, as any slight change of weights will lead to the change of the whole cost. In the other words, the whole cost forms the force that propels the change of weights. This could cause the whole cost fast converge to a local minimum. However, the training factor γ should be very small in order to make the cost converge. In our experiments, γ is set to 0.00001.

IV. Experiments

A. The artificial time-series data

In order to test the model described above, an artificial time series is created from the non-linear equation (10).

$$x_{t+1} = ax_{t-2}^2 + bx_t + \epsilon_t \quad (10)$$

Obviously, the non-linear regressor order of this time series is 2 (it is generated from two past values). We should note the lack of the x_{t-1} term, as well as the presence of the noise ϵ_t. The series does not contain any exogenous variable. Fig. 2 shows a time series with 1000 points obtained by Equation (10) with $a = -0.8, b = 0.9$ and the noise $\epsilon_t = rand() * 0.1 * max(time\_series)$. We sample the time series to produce two sets of data. One is a set of three dimension data with two input variables x_{t-2} and x_t and one dependent variable x_{t+1}; the other is a set of four dimension data with additional input variable x_{t-1}. We will examine the performance of the model for the two cases.

The steps of the experiment are described as below:

- produce k clusters with the k-hyperplane clustering algorithm, each of which corresponds to a hyperplane represented by a linear function;

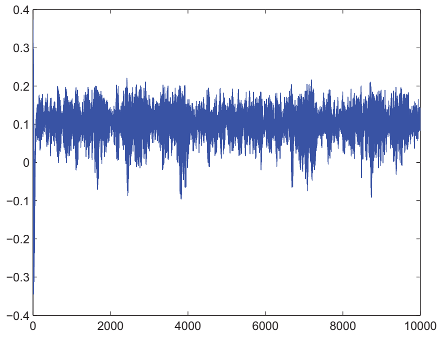


Fig. 2. A time series with 1000 points

- estimate the training set with the k linear functions, and thus each sample will have k estimates, which will be inputs of the neural network;
- run the gradient training algorithm to train the neural network, so that a set of weights can be obtained;
- estimate the test set with k linear functions;
- use the trained neural network to fuse the the k groups of estimates of the test data;
- evaluate the prediction accuracy through calculating the mean squared errors of the estimates to the real values, and the maximum errors for all test data.

In the experiments, the parameter k is fixed to 8. The mean squared error is calculated across the test database:

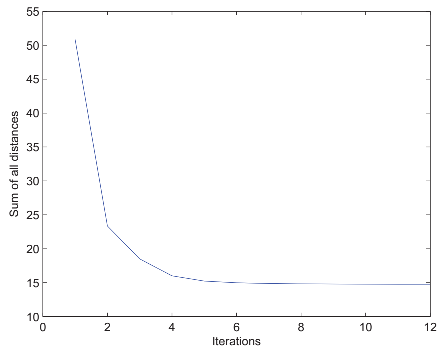
$$MSE = \frac{1}{|DB|}\sum_{i \in DB}(y(i) - \tilde{y}(i))^2. \quad (11)$$

B. Results on the three dimension data set

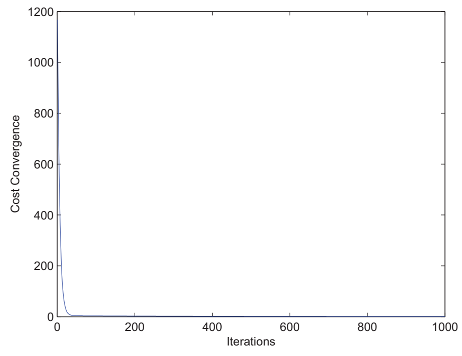
The first experiment is done on the three dimension samples with two input variables x_{t-2} and x_t, through which we predict the dependent variable x_{t+1}. When running the k-hyperplane clustering algorithm on the three-dimension training data, the termination criterium (the positive threshold α) is set to 0.05. It indicates that the iteration will be terminated when the difference between the sum of distances for current k clusters and that for last k clusters is less than 0.05. If the value of α is too small, when the cost be convergent to a certain value, it arrives at an oscillation state. Fig. 3 (a) displays the convergence process of the clustering algorithm for the three dimension data. The convergence speed is very fast, and by the 6-th iteration, the cost almost stops the decreasing. Finally, the sum of all distances converges to around 15. The process goes only for 12 iterations.

We also observe the convergence of the neural network when training it. The cost almost drops to zero from around 1200 in 40 iterations, and the dynamic system arrives at a local optimum (see Fig. 3 (b)).

Now let's see the prediction accuracy for the three dimension data. The experimental results are presented in scatter plot with the predicted values as y-axis against real values as x-axis (see Fig. 4). All error-free predictions should fall on



(a) k-hyperplane algorithm convergence



(b) Neural network training convergence

Fig. 3. convergence for 3-dim data

the line $y = x$. The mean squared error is 0.00018695, and the maximum error is 0.0849. It can be seen the prediction values fit the line $y = x$ very well.

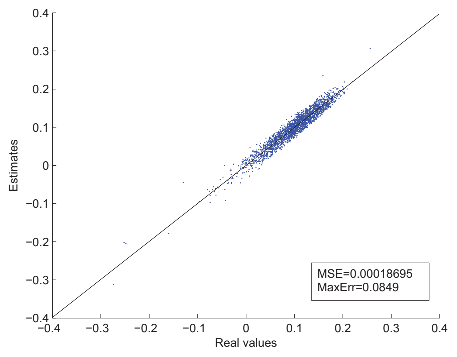
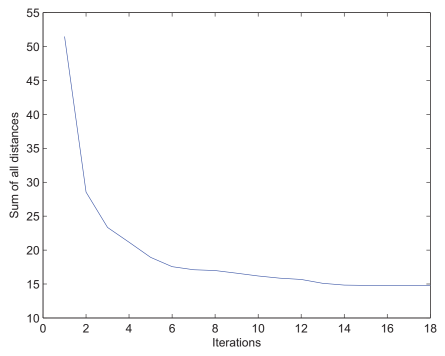


Fig. 4. Scattering the estimates against the real values for 3-dim data

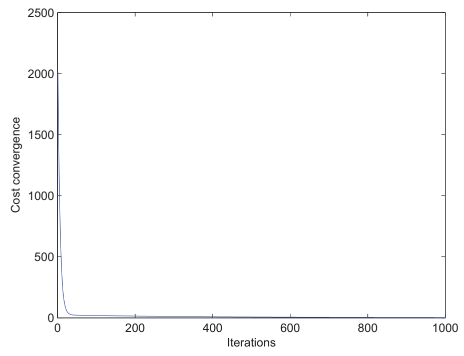
C. Results on the four dimension data set

The second experiment is done on the four dimension samples with Three input variable x_{t-2}, x_{t-1} and x_t, through which we predict the dependent variable x_{t+1}. The parameters of algorithms in this experiments are the same as that in the first experiment. Fig. 5 (a) displays the convergence process of k-hyperplane clustering algorithm for the four dimension data. It can be seen that the sum of distances arrives at the stable state of around 15, which is similar to the final convergence state in the first experiment. However, the convergence is not as fast as that in the first experiments. It may be because the variable x_{t-1} is added as an input variable, which change the mapping function to be with three independent variables. However, actually when we produce the time series, we only use two independent variables x_{t-2} and x_t. The process goes for 18 iterations.

From the Fig. 5(b), it can be seen that the convergence of the trained neural network is similar with the process in the first experiments. The initial cost for the results from k linear functions is around 2000. Obviously, the results from the k linear functions are worse than that in the first experiments. However, the neural network fusing the results from k hyperplanes can obtain approximate same performance for the two test data sets. The mean squared error is 0.00018531, and the maximum error is 0.0672. Fig. 6 shows that the scatter plot of estimates against real values fits the $x = y$ line very well.



(a) k-hyperplane algorithm convergence



(b) Neural network training convergence

Fig. 5. convergence for 4-dim data

V. Conclusions

In summary, we present a novel k-hyperplane based neural network model for non-linear regression. The experiments are done on an artificial time series, and the performance of the model is observed for two cases, three dimension data and four

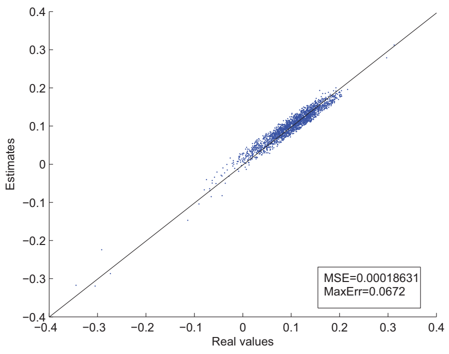


Fig. 6. Scattering the estimates against the real values for 4-dim data

dimension data, respectively. It is shown that the dimension of inputs affect the clustering performance very much. Neural network fusion can get some compensation in performance. The prediction performance is very good for the artificial time series. The further work will concern with the dynamic size of clusters, and adjustment of neural network parameters and structures. We will further exploit our approach for prediction problems in finance, market or environment areas.

REFERENCES

[1] L.H. Chen and C.C. Hsueh, Fuzzy regression models using least-squares method based on the concept of distance, *IEEE Transaction on Fuzzy Systems*, (2009), in press.

[2] M.M. Nasrabadi and E. Nasrabadi, A methematical-programming approach to fuzzy linear regression analysis, *Applied Mathematics and Computation*, **155**, (2004), pp. 873-881.

[3] H.C. Wu, Linear regression analysis for fuzzy input and output data using the extension principle, *Computers and Mathematics with Applicaitons*, **45**, (2003), pp. 1849-1859.

[4] C,Kao, C.L. Chyu, A fuzzy linear regression model with better explanatory power, *Fuzzy Sets and Systems*, **126**, (2002), pp. 401-409.

[5] K.K. Yen, S. Ghoshray and G. Roig, A linear regression model using triangular fuzzy number coefficients, *Fuzzy Sets and Systems*, **106**, (1999), pp. 167-177.

[6] B. Kim and R.R.Bishu, Evaluation of fuzzy linear regression models by comparing membership funcitons, *Fuzzy Sets and Systems*, **100**, (1998), pp. 343-352.

[7] H. Tanaka and H. Lee, Interval regression analysis by quadratic programming approach, *IEEE Transactions on Fuzzy Systems*,**6(4)**, (1998), pp. 473-481.

[8] G. Peters, Fuzzy linear regression with fuzzy intervals, *Fuzzy Sets and Systems*,**63**, (1994), pp. 45-55.

[9] D. T. Reden and W.H. Woodall, Properties of certain fuzzy linear regression methods, *Fuzzy sets and Systems*, **66**, (1994), pp. 361-375.

[10] M. Sakawa and H. Yano, Multiobjective fuzzy linear regression analysis for fuzzy input-output data, *Fuzzy Sets and Systems*, **47**, pp. 173-181.

[11] D. Savic and W. Pedryez, Evaluaiton of fuzzy linear regression models, *Fuzzy Sets and Systems*, **39**, (1991), pp. 51-63.

[12] H. Tanaka and J. Watada, Possibilistic linear systems and their application to the Linear regression models, *Fuzzy Set and Systems*, **27**, (1988), pp. 275-289.

[13] H. Tanaka, S. Uegima, K. Asai, Linear regression analysis with fuzzy models, *IEEE Transactions on System, Man and Cybernetics*, **12**, (1982), pp. 903-907.

[14] C,Kao, C.L. Chyu, Least-squares estimates in fuzzy regression anaysis, *European Journal of Operational Research*, **148**, (2003), pp. 426-435.

[15] B. Wu and N.F. Tseng, A new approach to fuzzy regression models with application to business cycle analysis, *Fuzzy Sets and Systems*, **130**, (2002), pp. 33-42.

[16] R. Körner and W. Nẗher, Linear regression with random fuzzy variables: extended classical estimates, best linear estimates, least squares estimates, *Informaiton Science*, **109**, (1998), pp. 95-118.

[17] P. Diamond and R. Körner, Extended fuzzy linear models and least squares estimates, *Computers and Mathematics with Applicaitons*, **33**, (1997), pp. 15-32.

[18] M. Ma, M. Friedman and A. Kandel, General fuzzy least quares, *Fuzzy Sets and Systems*, **88**, (1997), pp. 107-118.

[19] P.T. Chang and E.S. Lee, A generalized fuzzy weighted least-squares regression, *Fuzzy Sets and Systems*, **82**, (1996), pp. 289-298.

[20] P. Diamond, Fuzzy least squares, *Information Science*, **46**, (1988), pp.141-157.

[21] C.K. Kwong, Y.Chen, K.Y. Chan and H. Wong, The hybrid fuzzy least-squares regression approach to modelling manufacturing processes, *IEEE Transaction on Fuzzy Sysytems*, **13(6)**, (2008), pp. 644-651.

[22] P.Y. Hao and J.H. Chiang, Fuzzy regression analysis by support vector learning approach, *IEEE Transaction on Fuzzy Systems*, **16(2)**, (2008), pp. 428-441.

[23] D.H. Hong and C. Hwang, Interval regression anaysis using quadratic loss support vector machine, *IEEE Transaction on Fuzzy Systems*, **13(2)**, (2005), pp. 229-237.

[24] D. Zhang, L.F. Deng, K.Y. Cai and A. So, Fuzzy nonlinear regression with fuzzified radial basis function network, *IEEE Transacitons on Fuzzy Systems*,**13(6)**, (2005), pp.742-760.

[25] R.J. Hathaway and J.C. Bezdek, Switching regression models and fuzzy clustering, *IEEE Transactions on Fuzzy Systems*, **1(3)**, August (1993), pp. 195-204.

[26] M.S. Yang and C.H. Ko, On Cluster-Wise Fuzzy Regression Analysis, *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, **27(1)**, Feb. (1997), pp. 1-13.

[27] Takefuji, Y. and Szu H., Design of parallel distributed Cauchy machines [J], in *Proc of IJCNN Internal Joint Conference on Neural Networks*, (1989), pp. 529-532.